

LECTURE I: DATA STRUCTURE AND INTRODUCTION TO PYTHON

SEHH2239 Data Structures

LEARNING OBJECTIVES:

- To overview the significance of data structure
- To review the elements in a python program
- To review the expressions, operators, and flow control in python programs
- To declare list structure and access list elements

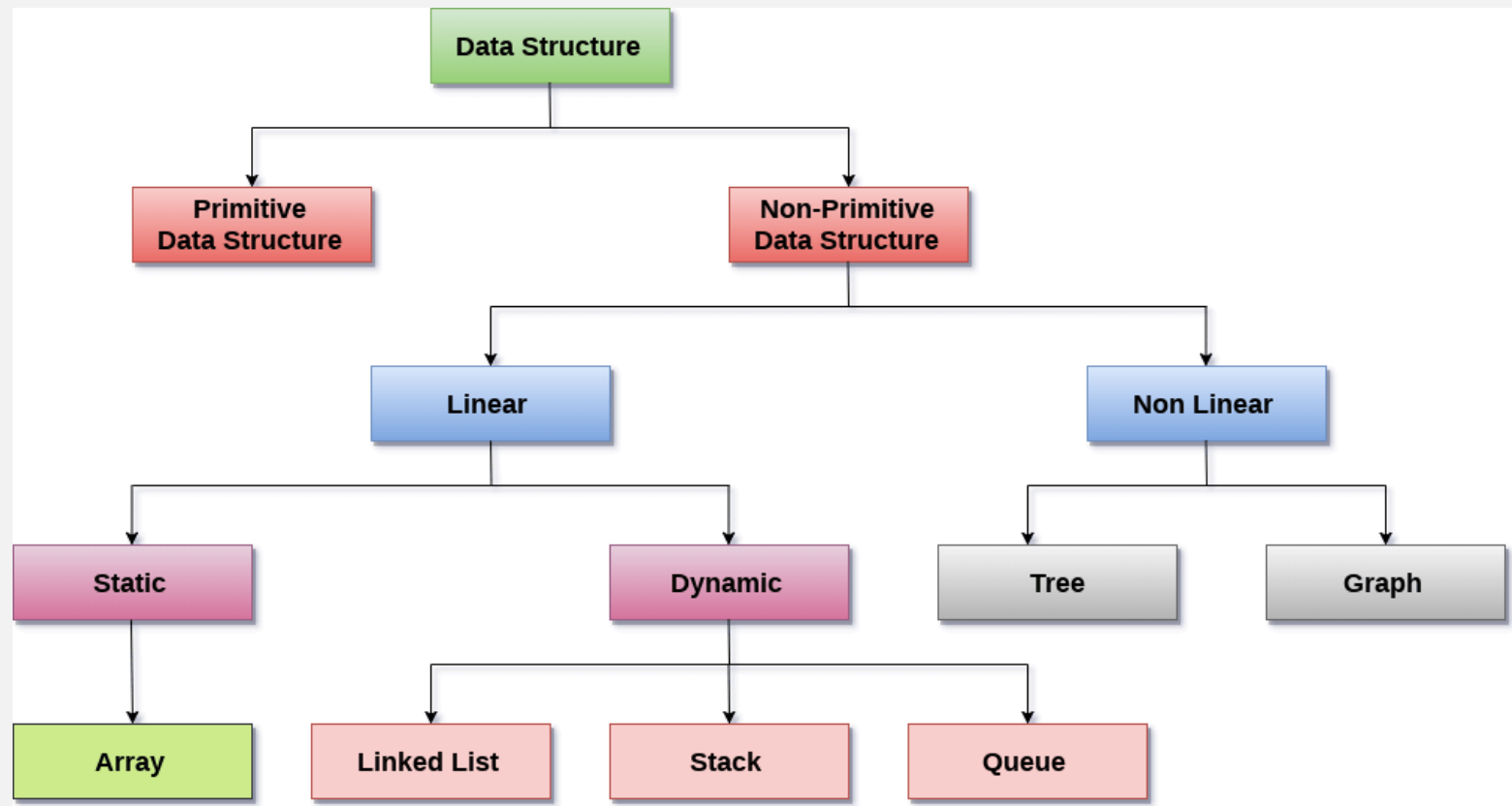
DATA STRUCTURE

- **Data**
 - can be defined as a representation of facts, concepts, or instructions in a formalized manner.
 - Data is represented with the help of characters such as alphabets (A-Z, a-z), digits (0-9) or special characters (+,-,/,*,<,>,<= etc.)
- A **data structure** is a specialized format for
 - organizing data,
 - processing data,
 - retrieving data and
 - storing data.
- In computer programming, a data structure may be selected or designed to **store data** for the purpose of **working on it** with various **algorithms**.
- Each data structure contains information about the data *values, relationships between the data* and *functions* that can be applied to the data.

IMPORTANCE OF DATA STRUCTURES

- Data structures are essential for **managing large amounts of data**, such as information kept in databases or indexing services, efficiently.
- Proper maintenance of data systems requires the **identification of memory allocation, data interrelationships and data processes**, all of which data structures help with.
- Choosing an ill-suited data structure could result **in slow runtimes or unresponsive code**.
- A few factors to consider when picking a data structure include
 - what **kind** of information will be stored,
 - **where** should existing data be placed,
 - how should data be **sorted** and
 - **how much memory** should be reserved for the data

DATA STRUCTURE CLASSIFICATION



OVERVIEW OF TYPES OF DATA STRUCTURES

Primitive and non-primitive

- **Primitive** are predefined types of data supported by the programming language.
 - For example, integers, float numbers, and characters are all primitive data types. E.g. 3, 0.45, s, ^,...
- **Non-primitive** are not defined by the programming language but are instead created by the programmer.
 - They are sometimes called "reference variables" or "object references" since they reference a memory location, which stores the data.

INTRODUCTION TO PYTHON

WHAT IS PYTHON?

- Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
- It is used for:
 - web development (server-side),
 - software development,
 - mathematics,
 - system scripting.
- What can Python do?
 - Python can be used on a server to create web applications.
 - Python can connect to database systems. It can also read and modify files.
 - Python can be used to handle big data and perform complex mathematics.
 - Python can be used for rapid prototyping, or for production-ready software development.

HELLO WORLD IN PYTHON

- In Google colab
 - <https://colab.research.google.com/notebooks/>


```
print("Hello World!")
```

- In IDE (e.g. PyCharm)
 - Save it as helloWorld.py

PYTHON EXPRESSIONS AND OPERATORS

COMPONENTS OF A PYTHON PROGRAM

- Executable statements are placed in **functions**
- Known as **methods**, that belong to class definitions.



```
class Person:
    def __init__(self, name):
        self.name = name

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("John")
p1.myfunc()
```

```
>>Hello my name is John
```

EXPRESSIONS AND OPERATORS

- The semantics of an **operator** depends upon the type of its **operands**.
- For example
 1. a and b are **numbers**, the syntax $a + b$ indicates **addition**
 2. a and b are **strings**, the *operator* $+$ indicates **concatenation**
- **Assignment Operator (=)**
 - = is used in python to assign values to variables

NUMBERS IN PYTHON

- Different numeric types in Python:

- **Int**

- `x = 5`

- **Float**

- `y = 3.46`

- **Complex**

- `z = 2 + 5j`



To view the number type:

```
print(type(x))  
>><class 'int'>
```

```
print(type(y))  
>><class 'float'>
```

```
print(type(z))  
>><class 'complex'>
```

- **Scientific numbers**

- `m = 2e5`

- `n = 3.53E4`



To print the number:

```
print(m)  
>> 200000.0
```

```
print(n)  
>> 35300.0
```

STRINGS

- Both `""` or `' '` can be used to quote strings
 - `a="ABC"`
 - `b1='dddggg lll llll .adf'`
 - `c_dfasdf_='123123'`
 - `d="'swing swimming so many medals add oil!'"`

To print the string

```
print(a[2])  
>> c
```

```
print(d)  
>>'swing swimming so many medals add oil!'
```

STRINGS

- Get a character from a string

```
print(a[2])  
>>C
```

- String length

```
print(len(a))  
>>3  
print(len(b1))  
>>20
```

- Slicing

```
print(b1[3:8])  
>>ggg l
```

- Search in a string

```
print("silver" in d)  
>>False
```

```
if "medals" in d:  
    print("yes, found")  
>>yes, found
```

ESCAPE CHARACTERS

- Allows you to use some special characters

```
print("\t\"hello world\")
```

```
>> "hello world"
```

Escape Sequence	Meaning
\newline	Backslash and newline ignored
\\	Backslash (\)
\'	Single quote (')
\"	Double quote (")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\v	ASCII Vertical Tab (VT)
\ooo	Character with octal value <i>ooo</i>
\xhh	Character with hex value <i>hh</i>

CASTING OF NUMBERS

- Casting in python is done by **constructor** functions:
- **int()** - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- **float()** - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- **str()** - constructs a string from a wide variety of data types, including strings, integer literals and float literals

```
x = 5.5  
y = int(x)  
print(y)
```

>>5

ARITHMETIC OPERATORS

- Python supports the following arithmetic operators:

+	addition
−	subtraction
*	multiplication
/	division
%	the modulo operator

- If both operands have type int, then the result is an int
- If one or both operands have type float, the result is a float.
- Integer division has its result truncated.

Try this: operators.ipynb

https://www.w3schools.com/python/python_operators.asp

INCREMENT AND DECREMENT OPS

- Python does not have unary increment/decrement operator (++/--). Instead to increment a value, use

```
a=2  
a += 1  
print(a)
```

```
>>3
```

LOGICAL OPERATORS

- Python supports the following operators for numerical values, which result in *Boolean values*:

- `x < y`
- `S == "abc"`
- `b1 != true`

<code><</code>	less than
<code><=</code>	less than or equal to
<code>==</code>	equal to
<code>!=</code>	not equal to
<code>>=</code>	greater than or equal to
<code>></code>	greater than

- Boolean values also have the following operators:

- `x < y && S == "abc"`
- `!(x < y)`
- `(S == "abc" || b1 != true)`

<code>!</code>	not (prefix)
<code>&&</code>	conditional and
<code> </code>	conditional or

PYTHON INDENTATION

- Indentation refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.

```
y = 14
if y<5:
    print("this line run when true")
print("this line run anyway")
```

```
>>this line run anyway
```

FLOW CONTROL

IF STATEMENTS

- The syntax of a simple **if** statement is as follows:

```
x=5
y=13
if y>x:
    print("y is greater.")
```

>>y is greater.

- if...else...**

```
y=2
if y>x:
    print("y is greater.")
else:
    print("y is not greater.")
```

>>y is not greater.

COMPOUND IF

- There is also a way to group a number of boolean tests, as follows:

```
if firstBooleanExp:
    firstBody
elif secondBooleanExp:
    secondBody
else:
    thirdBody
```

```
if y>x:
    print("y is greater.")
elif y==x:
    print("they are equal.")
else:
    print("y is smaller.")
```

- Short-hand if...else (Ternary Operator)

```
variable = expressionTrue if condition else expressionFalse;
```

```
# Python Ternary Operator
ylarge = True if y >x else False
```


WHILE LOOPS

- Such a loop tests that a certain condition is satisfied and will perform the body of the loop each time this condition is evaluated to be true.
- The syntax for such a conditional test before a loop body is executed is as follows:

```
while booleanExpression:  
    loopBody
```

```
i=1  
while i<6:  
    print(i)  
    i+=1
```

```
>>  
1  
2  
3  
4  
5
```

BREAK AND CONTINUE

- **break** statement that immediately terminate a while or for loop when executed within its body.

```
a = 3
while a < 7:
    print(a)
    if a == 4:
        break
    a += 1
```

>> 3
>> 4

- **continue** statement that causes the current iteration of a loop body to stop, but with subsequent passes of the loop proceeding as expected.

```
a = 3
while a < 7:
    a += 1
    if a == 5:
        continue
    print(a)
```

>> 4
>> 6
>> 7

FOR LOOPS

- The traditional **for-loop** syntax consists of four sections:
 - an initialization
 - a boolean condition
 - an increment statement
 - and the body—although any of those can be empty.
- The basic structure is as follows:

```
for counter in range(m, n)  
    loopBody
```

```
for i in range(0, 4):  
    print(i)
```

```
>>  
0  
1  
2  
3
```

VARIABLE NAMES IN PYTHON

- Rules for Python variables:
 - A variable name must start with a letter or the underscore character
 - A variable name cannot start with a number
 - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
 - Variable names are case-sensitive (age, Age and AGE are three different variables)

KEYWORDS IN PYTHON

- Keywords in Python are reserved words that cannot be used as a variable name.

```
help("keywords")
```

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

SUMMARY OF KEY TERMS

- Data Structure
- Basic Python
 - Expressions and Operators
 - Arithmetic
 - Logical
 - Relational
 - Flow Control
 - If
 - while/do while / for

REFERENCES:

- Textbook:
 - **Adam Drozdek, Data Structures and Algorithms in Python , 1st Edition, Cengage Learning, 2021**
- References
 - https://www.tutorialspoint.com/computer_fundamentals/computer_data.htm
 - <https://searchsqlserver.techtarget.com/definition/data-structure>
 - <https://www.w3schools.com/python/>
 - <https://www.geeksforgeeks.org/array-copying-in-python/>